National Defence / Défense nationale

A FREQUENCY HOPPING SYNTHESIZER CONTROLLER FOR EHF SATCOM

by

Claude Brochu
EHF SATCOM Section
Electronics Division

DEFENCE RESEARCH ESTABLISHMENT OTTAWA
TECHNICAL NOTE 89-29

PCN
D6470-0459B

July 1989
Ottawa

ABSTRACT

The Ground Terminal Simulator is one of the in-house test facilities being developed for the system and technology evaluation phase of the **FASSET** project. A synthesizer controller has been developed using a personal computer to test the frequency hopping subsystem of the Ground Terminal Simulator. Hardware and software components of the test system are described at different levels of complexity: the level required for a user to utilize the system and the technical level required to understand and modify the hardware and to upgrade the software.

RESUME

Le simulateur de station terreste de satellite est une des installations internes de tests en cours de développement pour la phase d'évaluation de technologie et de système de **FASSET**. Un système de commande pour un synthétiseur de fréquence a été développé pour faire la vérification du sous-système de bond de fréquence du simulateur. Le matériel et le logiciel constituant ce système de vérification sont décrits à différents niveaux de complexités soit le niveau requis pour un utilisateur et celui requis pour maintenir et améliorer le matériel et le logiciel.

EXECUTIVE SUMMARY

The development of a Ground Terminal Simulator supports the goals of the FASSET project of building an extended technology base within DND and expanding in-house expertise in the EHF military SATCOM field. A critical part of the simulator is the frequency hopping sub-system. To test this sub-system a synthesizer controller was developed. This technical note describes the components of the synthesizer controller, the assembly and operation of it, and the areas where improvement is possible.

The frequency hopping synthesizer controller is composed of a high performance Personal Computer, a digital I/O board, an interface board for the COMSTRON synthesizer, and software required to perform testing of the synthesizer in a number of modes.

A performance analysis was done on the controller and the Comstron synthesizer operating in the frequency hopping mode. It demonstrated that the Personal Computer could not support real time operation because of speed constraints. However, even with this restriction it will be of value in testing the frequency hopping subsystem in the construction stage of the ground terminal simulator.

TABLE OF CONTENTS

ABSTRACT iii

RESUME iii

EXECUTIVE SUMMARY v

TABLE OF CONTENTS vii

LIST OF FIGURES x

LIST OF ABBREVIATIONS xi

1.0 INTRODUCTION 1

2.0 CONTROLLER OPERATION 2
- 2.1 EQUIPMENT 2
- 2.2 CONTROLLER ASSEMBLING 2
- 2.3 COMPUTER SETUP 3
- 2.4 USE OF THE PROGRAM 4
  - 2.4.1 SINGLE FREQUENCY MODE OF OPERATION -- Mode 1 4
  - 2.4.2 STEP FREQUENCY MODE OF OPERATION -- Mode 2 4
  - 2.4.3 FREQUENCY HOPPED MODES OF OPERATION 5
    - 2.4.3.1 Frequency Hopping with Fixed Table Values -- Mode 3 5
    - 2.4.3.2 Frequency Hopping with Specific Values -- Mode 4 6
    - 2.4.3.3 Specific Value Disk File Format 6
  - 2.4.4 EXIT -- Mode 5 7

3.0 TECHNICAL DOCUMENTATION 8
- 3.1 DIGITAL I/O BOARD 8
  - 3.1.1 THE 8255A PROGRAMMABLE PERIPHERAL INTERFACE IC 8
  - 3.1.2 BASE BOARD PROGRAMMING 9
- 3.2 FREQUENCY SYNTHESIZER 10
  - 3.2.1 INTERFACING AND PROGRAMMING THE SYNTHESIZER 10
  - 3.2.2 SYNTHESIZER INTERFACE 11
- 3.3 FREQUENCY HOPPING CONTROL BY INTERRUPT PROCESSING 12
  - 3.3.1 INTERRUPT SYSTEM ON A PC 14
  - 3.3.2 FREQUENCY HOPPING CONTROL PERFORMANCE 15

4.0 PROGRAM DESCRIPTION 17
4.1 MAIN PROGRAM -- SYNTCTR 18
4.1.1 SINGLE FREQUENCY MODE 18
4.1.2 STEP FREQUENCY MODE 18
4.1.3 FREQUENCY HOPPING MODES 18
4.1.4 EXIT 19
4.2 MAIN SUBROUTINES 19
4.2.1 SINGLE FREQUENCY -- SNGFRQ 19
4.2.2 MULTIPLE FREQUENCY -- MLTFRQ 19
4.2.3 FREQUENCY HOPPING INTERRUPT CONTROLLED 20
4.2.3.1 Routine FHINT 20
4.2.3.2 Routine SUINT 20
4.2.3.3 Interrupt Service Routine 21
4.2.3.4 Interrupt Utility Routines in Module SUINT . . 21
4.3 SECONDARY SUBROUTINES 22
4.3.1 DOUT 22
4.3.2 KBCHK 22
4.3.3 WAIT 22
4.4 PROGRAM MAINTENANCE 22

5.0 CONCLUSION 25

REFERENCES REF-1

APPENDIX A SOFTWARE SOURCE CODE LISTINGS A-1
A1 PROGRAM SYNTCTR A-1
A2 SUBROUTINE SNGFRQ A-5
A3 SUBROUTINE MLTFRQ A-8
A4 SUBROUTINE FHINT A-11
A5 SUBROUTINE SUINT A-14
A6 SUBROUTINE DOUT A-21
A7 SUBROUTINE KBCHK A-22
A8 SUBROUTINE WAIT A-24
A9 BATCH FILES LISTING A-25
A10 INCLUDE FILE LISTING A-26

APPENDIX B WIRE-WRAP CONNECTIONS FOR COMSTRON SYNTHESIZER CONTROL INTERFACE B-1

LIST OF FIGURES

Fig. 2.1 SYNTCTR program main menu 3
Fig. 2.2 Specific values disk file format example 7
Fig. 3.1 Programming example for the 8255A 9
Fig. 3.2 Synthesizer Controller block diagram. 11
Fig. 3.3 COMSTRON synthesizer interface schematic 13
Fig. 4.1 Software modules tree 17
Fig. 4.2 Filenames of software modules 23

LIST OF ABBREVIATIONS

DND	Department of National Defence
EHF	Extremely high frequency
EOI	End of interrupt
FASSET	Functional Advanced Development Model EHF SATCOM System for Evaluation and Test
FP	Floating point
IC	Integrated circuit
IMR	Interrupt mask register
I/O	Input / Output
IRQ	Interrupt request
ISR	Interrupt service routine
PC	Personal computer
PIC	Peripheral interrupt controller (8259A)
PPI	Programmable peripheral interface (8255A)
SATCOM	Satellite communications
SW	Switch

1.0 INTRODUCTION

In anticipation of future requirements for a Satellite Communication System (SATCOM) in the Extremely High Frequency (EHF) band, the Department of National Defence (DND) has sponsored a project for exploratory research and development within government and industry. The objectives of the project are to develop an extended technology base within DND, other government agencies, and Canadian industry to support and reduce the risk associated with an EHF military SATCOM System. These objectives will be accomplished through the contracted development of the Functional Advanced Development Model EHF SATCOM System for Evaluation and Test (FASSET) system, followed by a system and technology evaluation program undertaken primarily in government laboratories.

Included in the development of a technology base within DND is the fabrication of portions of an EHF SATCOM system. A critical part of the in-house ground terminal is the frequency hopping sub-system. To ensure that the sub-system performs correctly a synthesizer controller was assembled to test the sub-system. Described in this technical note are the hardware components, the software components and a performance analysis of the controller.

Section 2.0 is a description of the controller from a user's perspective. It includes a description of the hardware, of the installation of the interface boards in the PC, of the interconnecting of the equipment, and of the usage of the program. Section 3.0 is a technically oriented description of the hardware and software components in the controller. Section 4.0 is a description of all software modules at a level which could be used by a maintenance programmer. Section 5.0 presents the conclusions of the technical note with suggestions for improvement of the performance of the controller.

2.0 CONTROLLER OPERATION

This section describes how to use the synthesizer controller for testing of the frequency hopping operation of the ground terminal simulator. A brief description of the equipment to be interconnected is given followed by the installation instructions for the digital I/O and synthesizer interface cards inside the personal computer and the interconnection of the synthesizer and the interrupt clock cable to the PC. Finally a detailed explanation on running the computer program, the menu structure, and the menu selection is given.

2.1 EQUIPMENT

The components of the synthesizer controller are listed below:

a. an 80386 IBM compatible computer with an 80387 math coprocessor running under MS-DOS operating system;

b. a 96-bit parallel I/O card (the Base Board™);

c. an interface board to drive the synthesizer and map the BCD digits to the specific D-type 50-pin connector of the target synthesizer;

d. a Comstron synthesizer with a frequency range of 10 to 4000 MHz; and

e. the associated interconnecting cables.

The synthesizer interface board is piggy-backed on the Base Board and is connected to it by two 40-pin flat cables. A short length 50-pin flat cable and a BNC terminated coaxial cable are used for the external connections to the frequency synthesizer and the interrupt control system.

2.2 CONTROLLER ASSEMBLING

The Base Board and its attached synthesizer interface are installed in free slots in the PC. However, because the Base Board is in prototype form, 3 computer slots are required. The final version of the Base Board package will use only 2 computer slots.

before installation of the Base Board in the computer, a check of the interrupt source select switch on switch block SW3 of this board must be done. Parallel port 3, interrupt A is used by the system to interrupt the computer, so the user must make sure that SW3-4 is closed and that all other switches are open. The jumper header J1, on the Base Board, is already wired to connect the board interrupt output (the 8255A interrupt source) to the PC IRQ7 vectored interrupt line. That is, pin 7 of the jumper header is connected to pin 10 by

a short piece of soldered wire. The Base Board is then installed inside the PC using the installation procedure described in [1].

Two short cables, the synthesizer 50-pin flat cable and the interrupt clock coaxial cable extend from the back of the PC after installation. The connectors from the synthesizer to the PC must be properly oriented so that pin 1 of each cable are aligned.

2.3 COMPUTER SETUP

The personal computer runs under the MS-DOS operating system. After the computer is started, the DOS prompt is displayed on the screen. The user must change from the startup directory to the directory where the program and the data files are located and start the synthesizer Control Program. The following DOS commands will change the current directory to C:\GTS and start the program **SYNTCTR**. The user entry is underlined. The text which is not underlined is the computer prompt.

```
C:\> CD GTS
C:\GTS\> SYNTCTR
```

The screen is erased and the SYNTCTR main menu appears on the screen as depicted in Fig. 2.1. The program waits for a user input, an operating mode value.

```
D6470 - EHF GROUND TERMINAL PROJECT
Synthesizer Interface Tests

Mode    Operation

 1    Single Frequency
 2    Step   Frequency
 3    Frequency Hopped - Table Values
 4    Frequency Hopped - Specific Values
 5    Exit

 Mode:
```

Fig. 2.1 SYNTCTR program main menu

2.4 USE OF THE PROGRAM

The menu displayed in Fig. 2.1 shows the different modes of operation of the program. The modes are single frequency, step frequency, and multiple frequency operations (frequency hopping). A brief description of these modes are:

a. **single frequency mode** - one frequency value is generated continuously;

b. **step frequency mode** - one sweep of increasing frequencies is generated; and

c. **multiple frequency mode, (or frequency Hopped mode)** - the frequency values are taken from a table of frequencies. This table of frequencies may be generated internally by the program or read from a disk file stored in the current directory, **C:\GTS**.

In multiple frequency mode the user may select interrupt control for the frequency hopping rate. In the non-interrupt controlled operation the frequency hopping is free running and the rate depends on the computer speed.

2.4.1 SINGLE FREQUENCY MODE OF OPERATION -- Mode 1

In single frequency mode , the user is asked for a frequency value which is a number between 10,000,000 Hz and 3,999,999,999 Hz inclusively. The prompt is shown below.

```
Enter Single Frequency ( Neg. for Menu ) >
```

If a positive value in the range specified is entered the synthesizer outputs the selected frequency continuously. If a negative value is entered the program returns to the main menu. The program continually prompts for a new frequency until a negative number is entered.

2.4.2 STEP FREQUENCY MODE OF OPERATION -- Mode 2

In step frequency mode of operation, the frequency is swept once from a low to a high value. The frequency is increased by a fixed step until the highest value is reached. The length of each step (the dwell time) is input by the user. The prompt is shown below.

```
        Step Frequency Mode:  Please Enter 4 Numbers

Start ( Neg. for Menu ), Stop, Step Frequency and Dwell time (100th of sec)
?
```

The program request four numbers: the Start Frequency, Stop Frequency, Step Frequency and the Dwell Time. A negative start frequency will cause the program to return to the main menu. The frequency values are entered in Hz and

the dwell time in 100th sec., the dwell time resolution being .01 sec. The program continually prompts for new values until a negative start frequency is entered.

2.4.3 **FREQUENCY HOPPED MODES OF OPERATION -- Mode 3 & Mode 4**

In the frequency hopped modes, the frequency values are read from a table, converted from binary to BCD, and output to the synthesizer. The table is continuously scanned until interrupted by the user by entering any single character at the keyboard. There are two types of frequency tables implemented in this program. These are:

a. the table of frequencies is internally generated by the program, this is mode 3; or

b. the table of frequency is read from a disk file previously prepared, this is mode 4.

Alternatively there are two rates of scanning. These are:

a. the frequency table scanning could be free running at maximum computer speed; or,

b. the frequency table scanning could be at a fixed frequency hopping rate through an interrupt clock signal sent via the coaxial cable extending at the back of the computer.

2.4.3.1 **Frequency Hopping with Fixed Table Values -- Mode 3.** In mode 3 operation, the program displays the number of frequency values in its internal table and requests a table scan count from the user. The frequency scan is a frequency sweep of 1000 values ranging from 4 to 4000 MHz. This scan count is the number of times the table is scanned before a check is made for a user entered character to terminate the operation. The frequency table scanning continues if no character is entered.

The following messages and prompts are displayed on the screen:

Frequency Hopped Table Value

There are, 1000 Frequency Values in Table
Enter Table Scan Count > 1000

Any Character will stop Table Scan
Do you wish to use the interrupt mechanism (Y/N) ?

As seen above, the user is asked if the frequency hopping is to be controlled by an interrupt clock. A "no" answer will start the frequency hopping in a free running mode. The scanning will go as fast as the computer permits, and will continue uninterrupted until the user enters any character at the keyboard which will cause the frequency table scanning to stop and the main menu

to reappear. A "yes" answer to the interrupt query will send a message to the screen warning the user that the interrupt mechanism is on.

Interrupt Mechanism Set Up -- Interrupt ENABLED

When the user terminates the operation by entering a character at the keyboard, another message appears momentarily before the main menu returns to the screen.

Interrupt DISABLED -- Interrupt Mechanism Removed

The purpose of the scan count is to put a limit on the number of times the keyboard input buffer is checked. A large scan count will prevent too many interrupts of the frequency hopping to allow for the buffer content verification. As an example of a scan count value, assume a frequency hopping rate of 10 kHz and a table of 1000 frequencies, a scan count of 1000 will allow the frequency hopping interruption after 100 seconds or, conversely, every 100 seconds the keyboard input buffer is verified.

2.4.3.2 **Frequency Hopping with Specific Values -- Mode 4.** The only difference in mode 4 from mode 3 operation is that the program requests a File Name for the table of specific frequency values. Presently, the program supports up to 1000 frequency values stored on a disk file. Paragraph 2.4.3.3 below describes the format of the disk file.

The following message and prompt is sent when mode 4 is selected.

Frequency Hopped Specific Values

Enter Specific Values File Name > FHTBL.DAT

Then, as in mode 3, the number of frequency values in table is displayed and the requests for scan count and interrupt control are sent.

There are, 15 Frequency Values in Table
Enter Table Scan Count > 10000

Any Character will stop Table Scan
Do you wish to use the interrupt mechanism (Y/N) ?

2.4.3.3 **Specific Value Disk File Format.** The format of the disk file for the specific frequency values is described below. The first record contains a number N representing the number of frequency values stored in the subsequent records. The value of N is an (INTEGER) number between 1 and 1000. The (DOUBLE PRECISION REAL) frequency values are in Hz. Their maximum value as described in paragraph

2.4.1 above is 3,999,999,999 Hz. The user must note however that no commas are allowed inside a number in the file.

The numbers are read using List-Directed I/O. A list directed record is a sequence of values and value separators. Each value separator in a list-directed record must be either a comma (,), one or more contiguous blanks between two constants or after the last constant or an end of record mark. As the end of record mark in a file has the effect of a blank, the program will read as many records as required until all the N frequencies are read. Fig. 2.2 is an example of such a file.

```
15
20123456, 30987654, 40432765, 50864345, 60123876
70435876, 80123456, 90654321, 15000000, 25135790
35086421, 45246952, 55123456, 65000000, 75000000
```

Fig. 2.2 Specific value disk file format example

2.4.4 EXIT -- Mode 5

When Exit, mode 5, is selected the program terminates and the DOS prompt reappears as below.

```
C:\GTS>
```

3.0 TECHNICAL DOCUMENTATION

This section describes the components of the Synthesizer Control System. The digital I/O board (Base Board™) which provides parallel input and output lines, is described and its programming requirements are commented on. The frequency synthesizer is presented and a description of its interface to the Base Board is given. The personal computer interrupt processing mechanism is described along with the requirements for the interrupt service routine (ISR). Finally, a performance (timing) analysis of the test system operating in frequency hopping mode controlled by an interrupt clock is given.

3.1 DIGITAL I/O BOARD

The digital I/O board used for the synthesizer control system is the Scientific Solutions Inc. Base Board™. This board provides 96 digital input and output lines arranged in four sections of 24 lines each. Each section may be subdivided into three groups of 8 lines for mode 0 operation or into two groups of 12 lines for mode 1 operation. (These modes of operation are described below in 3.1.1). Each group of lines can be programmed for either input or output. A more detailed description may be found in [1].

The four 24-bit parallel ports are implemented with INTEL 8255A Programmable Peripheral Interface (PPI) parallel I/O ICs. Their output lines are connected to the outside world via four 40-pin connectors. The Base Board allows for customization by adding daughter modules that piggy-back onto the board. In this application an Amoeba board was used for the synthesizer interface construction. This board was designed for prototype wire-wrap circuit design.

3.1.1 THE 8255A PROGRAMMABLE PERIPHERAL INTERFACE IC

The 8255A PPI is a general purpose programmable I/O device, its 24 I/O lines form three 8-bit ports labelled A, B, and C. These 3 ports can be used in three modes of operation of which the first two modes, mode 0 and mode 1, have been implemented in the synthesizer control system. In mode 0, (Basic Input/Output), each 8-bit port is simply programmed for input or output. In mode 1, (Strobe Input/Output), each group of 12 bits or I/O lines, may be programmed to have eight lines of input or output and three for handshaking and interrupt control. The two remaining lines can be used independently for input or output. Mode 2, which is not used, is for strobed bidirectional bus I/O applications such as communications with a peripheral device on a single 8-bit bus for both transmitting and receiving data.

In the synthesizer control design, only two of the 4 sections of PPIs are used, section 1 and section 3. In non-interrupt processing the two PPIs are programmed in mode 0 parallel output, whereas in interrupt processing mode the second section, section 3, is programmed in mode 1 output. Two free lines (C4,C5) were programmed in output for strobing the synthesizer and to send an optional timing measurement pulse from the ISR to test point TP-1.

3.1.2 BASE BOARD PROGRAMMING

The Base Board is simple to program with the PC. One byte is sent to the 8255A control port to setup its mode of operation and then a byte of data is sent to each of its output ports A, B, and C. This programming is done in the Macro Assembler language using the OUT instruction to send an 8-bit data byte to an IO location. The Base Board occupies 16 I/O locations, four locations for each of the four parallel port sections. The Base Board programming documentation stored in the distribution diskette and [1] constitute good sources of programming information for the 8255A PPI. The example of Fig. 3.1 shows how to program section 1 IC in mode 0 to output 16 bits from variable BCDVAL to port A and B. BCDVAL is a variable that has previously received the BCD values to be output to the synthesizer.

```
   .data
   BCDVAL  DT    0                ;BCD data stored here

   .code
   CTRPRT1 EQU   0210H            ;Ctrl port 1 address
   PRTA    EQU   0211H            ;Port A address
   PRTB    EQU   0212H            ;Port B address

   MODE0   EQU   080H             ;Mode 0 output
   BCDVAD  EQU   BYTE PTR BCDVAL

;    SETUP FOR MODE 0

   MOV     AL,   MODE0
   MOV     DX,   CTRPRT1
   OUT     DX,   AL               ;Select Mode 0

;    OUTPUT TO PORT A FIRST BYTE OF DATA

   MOV     AL,   BCDVAD+[0]       ;load data byte
   MOV     DX,   PRTA             ;select port A
   OUT     DX,   AL               ;write data to port A

;    OUTPUT TO PORT B SECOND BYTE OF DATA

   MOV     AL,   BCDVAD+[1]
   MOV     DX,   PRTB
   OUT     DX,   AL
```

Fig. 3.1 Programming example for the 8255A

3.2 FREQUENCY SYNTHESIZER

The frequency synthesizer used for the internal FASSET ground terminal experiment is the COMSTRON FS2000-4. This synthesizer is a mid-range source operating over the range of 10 MHz to 4000 MHz with an output resolution of 1 Hz. It is capable of making a step frequency change from any point to any other point in less than 1 μsec. It incorporates a standard remote programming control capability which feature a 44-bit parallel BCD TTL positive true signal with TTL compatible strobe [2].

The synthesizer presents an output impedance of 50 ohms with an analog output amplitude of +10 dBm level, ±2 dB levelling. Each BCD input data line is equivalent to four TTL standard loads. A front panel display indicates the programmed frequency in GHz and except for an ON/STDBY switch, there are no local front panel frequency controls.

3.2.1 INTERFACING AND PROGRAMMING THE SYNTHESIZER

The synthesizer equipped with the standard parallel interface is designed for remote operation only. A suitable interface circuit must be provided in order to drive it. This interface must have the following characteristics:

a. the logic signals are TTL;

b. each line from the interface must be capable of driving 4 standard TTL loads;

c. the logic signals must be positive true, parallel BCD for a maximum of 10 digits with one strobe line. (38 + 1 lines are required); and

d. the input data are strobed on the trailing edge.

Programming of output frequency is accomplished by sending the appropriate BCD data, followed by a strobe, according to the timing requirements shown in the Synthesizer Operations Manual's, "Timing Diagram for Programming Input Signals" [2, Fig. 3-3]. It should be noted that the strobe allows for edge triggered latching of data for programming the synthesizer. The display will change to match the BCD data present at the programming input when the strobe is asserted high, but the synthesizer frequency will not change until a negative-going edge occurs.

The following connectors must be provided:

a. Two flat cable 40-pin connectors for attachment to the Base Board;

b. One flat cable 50-pin connector for the synthesizer cable; and

c. One BNC terminated coaxial cable for the interrupt hop clock signal.

More detailed information on programming and interfacing the synthesizer can be found in the Operation and Maintenance Manual [2].

3.2.2 SYNTHESIZER INTERFACE

An interface capable of driving the synthesizer's input lines was built to the interfacing requirements described in paragraph 3.2.1. The interface circuitry between the computer and the synthesizer consists of the line driver ICs, the two 40-pin connectors for interconnection to two of the Base Board's 8255A parallel I/O ICs, one 50-pin connector for the synthesizer connection, and one small coaxial cable BNC terminated for the external interrupt clock signal. A small length of flat cable was used for the interconnections. The synthesizer cable and the coaxial cable extend a few inches from the back of the computer. The synthesizer controller block diagram in Fig. 3.2 depicts the synthesizer controller components.

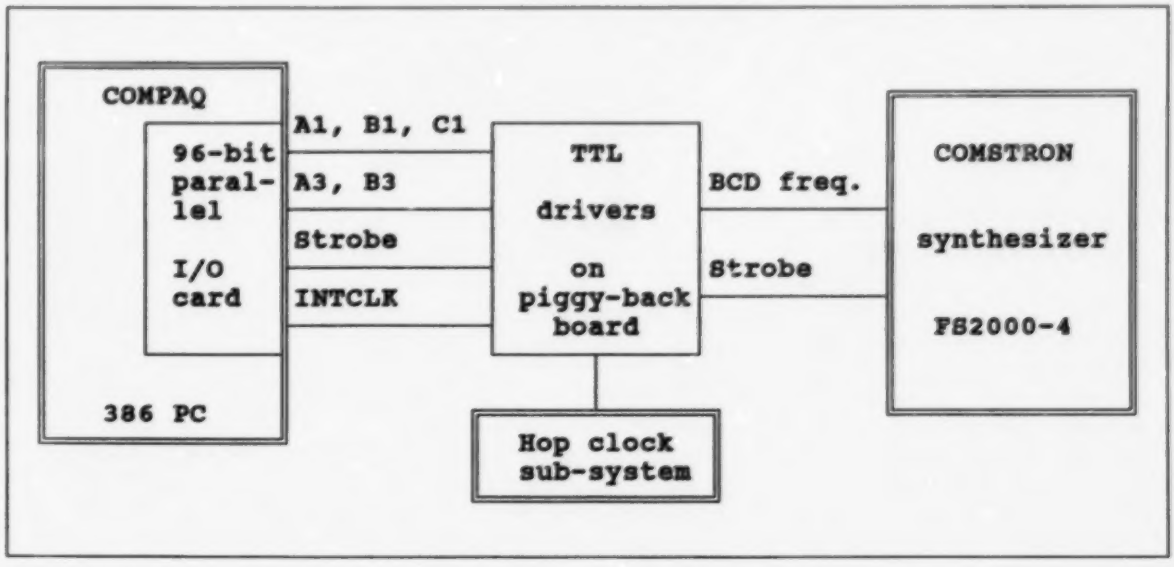

Fig. 3.2 Synthesizer controller block diagram

The line drivers chosen for the interface are the octal buffers and line drivers 74LS244 ICs with 3-state outputs. These devices feature high fan-out, good fan-in, and 400 mV. of noise margin. The 74LS244 have non-inverting outputs for positive true logic and symmetrical control inputs.

The synthesizer interface schematic of Fig. 3.3 shows the logic circuit interconnections. Five driver ICs are used for the synthesizer's 38 BCD frequency input lines or 10 decades from 1 Hz to 4 GHz. Two additional output lines are used to strobe the synthesizer and to, optionally, generate a timing

pulse at Test Point TP-1 to signify the end of interrupt service in the ISR. One of the interrupt handshaking lines, ACK_A, (pin C6 of section 3 PPI) is used to interrupt the computer via the INTCLK signal coming from the hop clock sub-system when the controller operates in interrupt mode. The hop clock signal originated from an HP pulse generator during the controller tests.

In Fig. 3.3, J1 and J2 are 40-pin connectors for connection to section 1 and section 3 PPI of the Base Board. J3 is the 50-pin connector for the COMSTRON synthesizer. U1 through U5 are the 74LS244 driver ICs. (Two 1 KΩ resistors to ground are connected to J3-30 and J3-37 for compatibility with the COMSTRON FS2000B synthesizer capable of higher frequency generation). J4 is a BNC connector at the end of the INTCLK short coaxial cable for connection to an external hop clock sub-system.

Three test points allow monitoring of the section 3 PPI interrupt signals. TP-2 is OBF_A, TP-3 is $INTR_A$ and TP-1 is the optional time mark used to signify the end of interrupt service in the ISR. TP-1 is one of the free output lines (pin C5) of section 3 PPI operating in mode 1, configuration 4 [1]. (The ISR must be modified and recompiled in order to produce this time mark. This operation will be covered in section 4.)

The synthesizer interface is built on the Amoeba board using wire-wrap connectors. This board is specially designed to piggy-back onto the Base Board. Using special care in the controller construction, the two boards package should not use more than 2 computer slots. A wire-wrap list of the interface board may be found in Appendix B.

3.3 FREQUENCY HOPPING CONTROL BY INTERRUPT PROCESSING

It is important that the frequency hopping rate be controlled so that the frequency is hopped at a known fixed rate and a set timing. A way to do it is to interrupt the computer via an external hardware interrupt signal (the hop clock signal) to start a hopping sequence at the required rate and timing. This sequence consist of strobing the synthesizer, generating the next frequency value, and waiting for the next hop clock signal. It is implemented in a routine invoked at interrupt time, the ISR.

The following paragraphs will describe the interrupt processing mechanism of the PC. Emphasis will be on the topics listed below:

a. the functions of the 8259A Programmable Interrupt Controller (PIC);

b. the modification of the interrupt mask register (IMR) for the particular Interrupt Request (IRQ) line to enable the interrupt capability of a particular device connected to that IRQ line;

c. the modification of an interrupt vector in the interrupt vector table to introduce the ISR to the interrupt system;

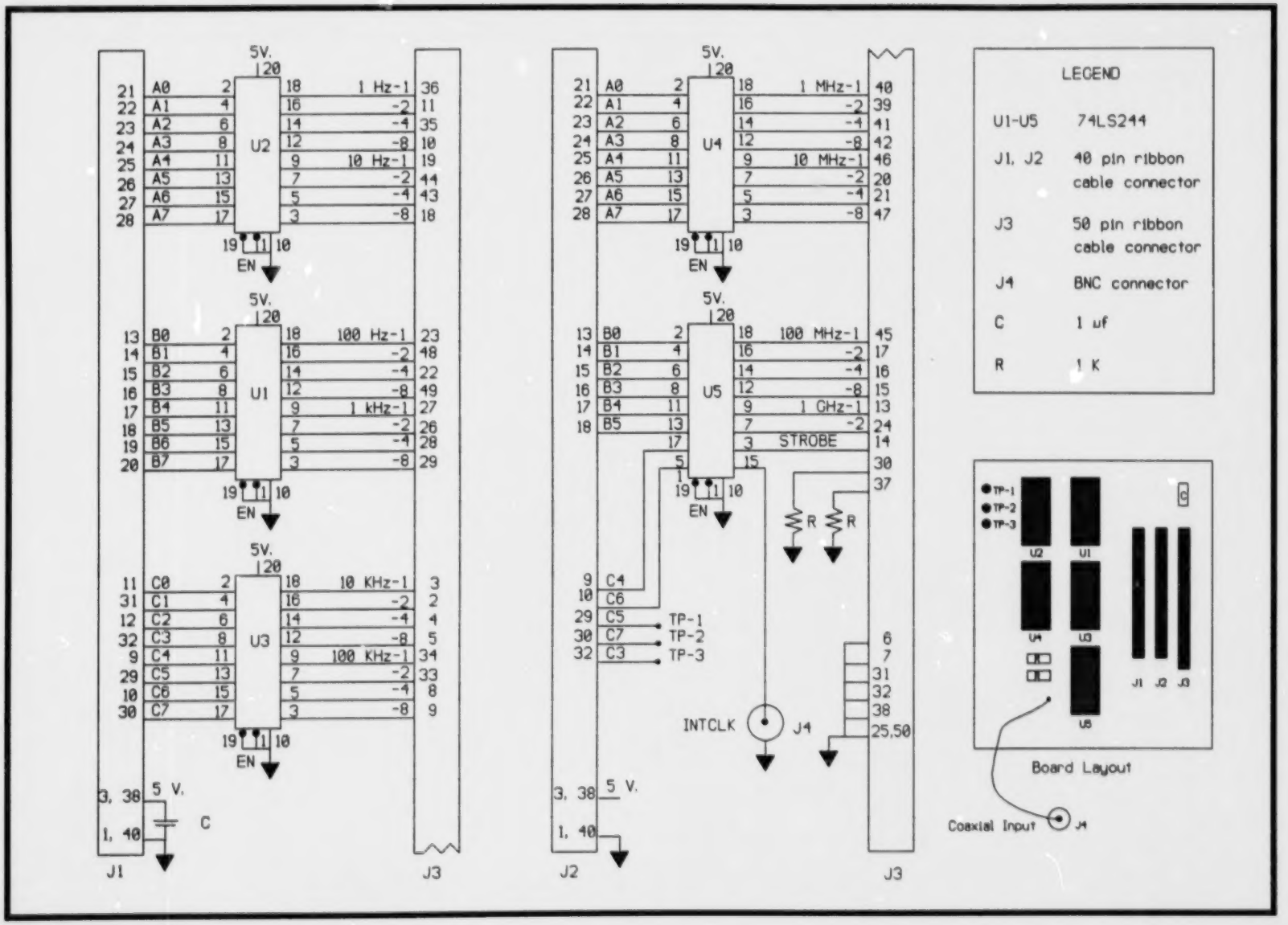

Fig. 3.3 Comstron synthesizer interface schematic

d. the start and end code of the ISR; and

e. the disabling of the interrupt when the program terminates, by resetting the interrupt vector table and IMR to their previous values.

3.3.1 INTERRUPT SYSTEM ON A PC

The 8086, 80286, 80386 family of microprocessors can handle different types of hardware interrupts such as the internal interrupts, the non-maskable interrupts, and the maskable interrupts. The maskable interrupts, which are of interest here, are usually generated by an external device and can be masked (ignored) through the IMR. They are routed to the main processor through the 8259A Programmable Interrupt Controller. When it receives an interrupt request, the PIC signals the microprocessor that an interrupt needs service.

The basic PC is designed to allow for eight maskable interrupts called IRQ0 through IRQ7. The IBM PC/AT and IBM compatibles which use an AT bus have an additional 8 levels of interrupt which are added by using a second cascaded 8259A. The IRQn lines are essentially "hardware" interrupt signals leading from peripherals that tell the system which peripheral wants to be heard. The IMR used to mask specific interrupts is internal to the PIC. When an IMR bit is set (at port 21H), the corresponding IRQ line is masked and no servicing for it is performed until the bit is clear.

When a device requires attention, it sends a signal to the PIC via its IRQ line. The PIC evaluates the service request, checking if this interrupt is masked off or other higher priority interrupts are being serviced, and if appropriate, interrupts the microprocessor. The microprocessor handles all interrupts by pushing the contents of the flag register onto the stack, disabling the interrupt flag, and pushing the current content of the instruction pointer register onto the stack. The microprocessor takes the interrupt number from the bus and processes it to find the address of the appropriate ISR in the interrupt vector table. The ISR is then started.

When interrupt processing is requested by a computer program, the mechanism must be setup by a special interrupt initialization routine. The following functions must be implemented in this routine:

a. the interrupt vector for the IRQ line the device is connected to is modified to identify the new ISR;

b. the old vector is saved for later restoring when the program terminates or interrupt processing is no longer required;

c. the IMR is modified to enable the device IRQ line to interrupt the microprocessor; and

d. a specific End of Interrupt (EOI) command is sent to the PIC to terminate any pending interrupt on that IRQ line that might still be active.

Another routine is also required to restore to their former values the old vector in the interrupt vector table and the interrupt mask register when the program terminates.

The first function listed above can be programmed by using the recommended MS-DOS functions provided for this purpose, interrupt 21H function 25H (set interrupt vector) and 35H (get interrupt vector). The PIC and the IMR have respectively a device code of 20H and 21H and only a simple OUT command is required to program them. The block below shows two ways the PIC may be programmed to specify an end of interrupt state and how to read and write onto the interrupt mask register.

```
OUT   20H,  67H        ;Issue specific EOI for IRQ7
OUT   20H,  20H        ;Issue non-specific EOI

IN    AL,   21H        ;Read in 8-bit register AL IMR content
OUT   21H,  AL         ;write from register AL to IMR
```

The interrupt service routine, which for programming convenience is part of the module containing the interrupt initialization routine, must preserve all microprocessor registers and conditions at entry. The interrupt is also usually reenabled (STI command) because the computer turns it off before giving control to the ISR. After the interrupt is serviced, the interrupt handler must accomplished the following exit functions:

a. interrupts are disabled (CLI command);

b. a non-specific EOI is signalled to the PIC;

c. registers and conditions at entry are restored; and

d. an exit from the ISR is done through an IRET command.

Examples of these techniques may be seen in APPENDIX A in the listing of module SUINT. A more in-depth treatment of the interrupt system is given in [3] and [4].

3.3.2 FREQUENCY HOPPING CONTROL PERFORMANCE

The main differences between the two frequency hopping modes of operation described above in section 2 is that the frequency hopping is either free running or controlled by an interrupting external clock signal.

Using a logic analyser to measure the logic signals timing on the interface board, it was found that the best frequency hopping rates obtainable were 50 μs per hop or less than 20 kHz for the free running modes, and 110 μs or less than 10 kHz for the interrupt controlled modes.

The difference in the time measurements results from the computer overhead required for interrupt processing. At interrupt time, the CPU must save the flags, disable interrupt and save the instruction pointer. The ISR must perform all handshaking and terminating action, reenable interrupt service, etc. More specifically the ISR must preserve all registers and conditions at entry and restore them before returning.

Other time measurements were performed during the execution of the program. The time durations between the hop clock pulse and the strobe pulse to the synthesizer were measured. The generation of the strobe pulse is the first instruction executed in the ISR after the environment save action at entry. The time after the BCD output to the synthesizer, which occurs just before the terminating action of the ISR, was also measured at TP-1 (see 3.2.2). These two time durations are 28 μs and 79.6 μs respectively. It is assumed the terminating action lasts about 28 μs, because what has been done in the interrupt service preamble must also been done at termination of interrupt. The total processing time of an interrupt is about 107.6 μs which is very close to the max rate of 110 μs observed.

More time measurements and timing comparisons led to the following approximative processing duration:

a. Floating point environment save, 18 μs;
b. Pre-interrupt action and Registers save, 10 μs;
c. Binary to BCD conversion, approx. 38 μs; and
d. BCD output via Base Board, approx. 15 μs.

It is evident that, when using the interrupt system to control the frequency hopping, it is not possible to hop as fast as in the free running mode. By looking at what the program does, means could be found to improve the frequency hopping rate performance. During this frequency hopping operation the program does nothing but looping and checking the keyboard input buffer, all the work is done by the ISR. Therefore it is not necessary to save the floating point environment in the ISR preamble. This will save about 18 μs in the total ISR duration. The hopping rate could then increase to a value less than 14 kHz or about 73 μs per hop. This constitutes an easy to implement improvement.

An additional mean of improving the performance is to convert the table of frequency values to BCD prior to the start of the hopping operation. The ISR would then output only the BCD values and another 38 μs will be saved. This could theoretically give a maximum hopping rate of less than 28 kHz. However this has to be checked because of the possible computer speed limitation. The implementation and test of these measures, to improve the performance of frequency hopping controlled by an external hop clock, could be the subject of future work on this project.

4.0 PROGRAM DESCRIPTION

This section is intended for the reader who will be responsible for maintaining the Synthesizer Control software. The computer used is a COMPAQ Deskpro 386 Personal Computer, it has an INTEL® 16-MHz 80386 microprocessor and a 16-MHz 80387 coprocessor, and runs under the COMPAQ MS-DOS Version 3.3 operating system. The programming languages used were Microsoft FORTRAN 4.1 and Macro Assembler 5.1. Most of the programming was done in FORTRAN except for the functions to write to the digital I/O board and the interrupt processing functions which were programmed in Macro Assembler.

The following paragraphs contains a description of the program module and subroutines of the synthesizer controller. The first module described is the main program **SYNTCTR**. The description of the main subroutines for single frequency, free running and interrupt controlled frequency hopping, and for setting up the interrupt processing follows along with an explanation of the interrupt utility routines. Finally, three other secondary subroutines for digital I/O, keyboard input buffer verification or for simply idling the program are described. Fig. 4.1 below shows the Software Module Tree of the synthesizer controller. The last paragraph deals with software maintenance.

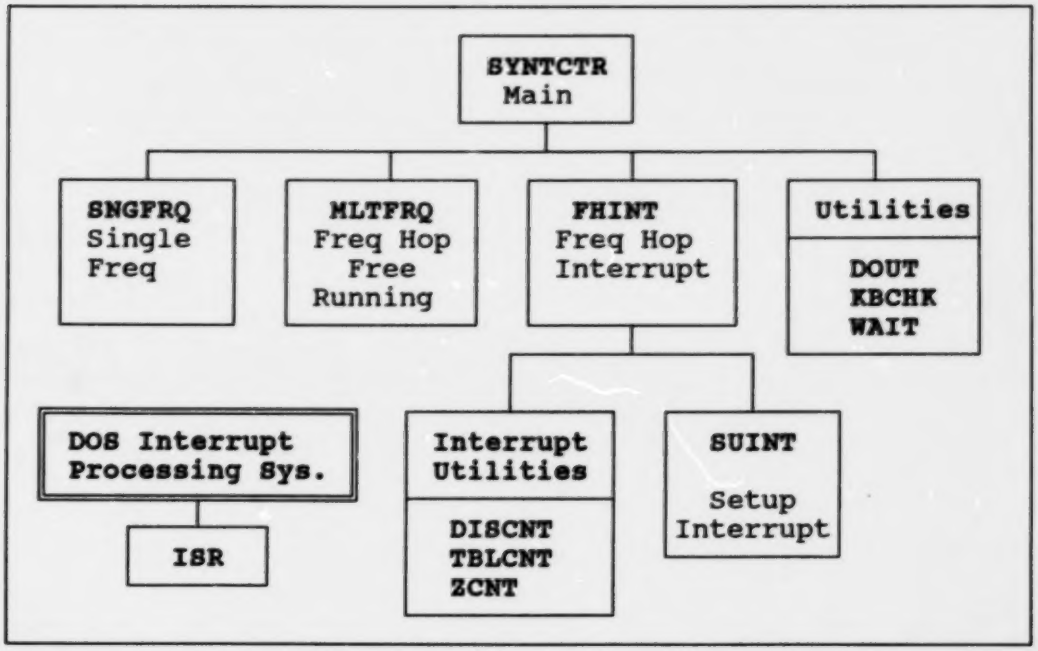

Fig. 4.1 Software modules tree

4.1 MAIN PROGRAM -- SYNTCTR

The program **SYNTCTR** is the main module of the Synthesizer Control program. This module, written in FORTRAN, controls the complete operation. The program initializes the digital I/O board and displays a menu on the screen requesting the mode of operation for frequency generation. After a selection is made by the user, the frequency data is prepared and the appropriate subroutine is called.

There are 5 modes of operation to choose from, (see the menu in Fig. 2.1):

a. single frequency;

b. step frequency;

c. frequency hopping from table values;

d. frequency hopping from specific values read from a disk file; and

e. exit from the program.

When using the frequency hopping modes it is also possible to control the frequency hopping rate and timing using interrupt processing via an external hop clock signal.

4.1.1 SINGLE FREQUENCY MODE

In single frequency mode, the user enters a frequency value and the assembly routine **SNGFRQ** is called to generate the BCD code and strobe the synthesizer. This mode stops when a negative value is entered. The prompt and input data format are described above in 2.4.1.

4.1.2 STEP FREQUENCY MODE

In the step frequency mode, the user enters the parameters required to generate the sweep. These are the four parameters Start, Stop, Step frequency and Dwell time described in 2.4.2. After data entry a loop is started in which the routines **SNGFRQ**, **WAIT** and **KBCHK** are called in sequence for each step of frequency. **WAIT** idles the program during the dwell time and **KBCHK** check the contents of the keyboard input buffer. The sweep continues until the last frequency or until a character (detected by **KBCHK**) is entered at the keyboard. New parameters may then be input for a new sweep or the mode may be stopped by entering a negative **start** value.

4.1.3 FREQUENCY HOPPING MODES

The two frequency hopping modes, mode 3 and mode 4, are similar. The frequency values are read from a table and repeatedly output to the synthesizer either at a free running rate or controlled by the hop clock sub-system. The difference between these two modes is in the way the frequency table is generated.

In mode 3 the frequencies come from an internal table of 1000 values and in mode 4 the values are read from a disk file stored in the current directory. This internal table of frequencies is described in 2.4.3.1 above. Paragraphs 2.4.3.2 and 2.4.3.3 describes how the disk file is input to the program along with its format.

The routine **MLTFRQ** is called for free running frequency hopping operation. The routine **FHINT** is called for clock controlled frequency hopping operation. As in the previous modes of operation the frequency table scanning stops when a character is entered at the keyboard.

4.1.4 EXIT

The program terminates when the user requests mode 5. No more processing is performed at this point, all hardware and software resets have been previously invoked by the lower level subroutines.

4.2 MAIN SUBROUTINES

4.2.1 SINGLE FREQUENCY -- SNGFRQ

The assembly routine **SNGFRQ** converts a binary double precision number into BCD and outputs the resulting BCD values to the synthesizer through the Base Board. The floating point (FP) instruction **FLD** loads the number into the first FP stack register and the instruction **FBSTP** converts the number into BCD and stores the results in the variable BCDVAL. The macro **OUTPUT** is then executed to output the BCD values to the synthesizer. The last command sent by **SNGFRQ** is the synthesizer strobe signal where bit C4 of section 3 8255A PPI is set/reset to generate the strobe pulse.

A macro is a mechanism of the Macro Assembler language that enables the assignment of a symbolic name to a block of source statements. At assembly time when the assembler encounters the text associated with a macro name, it replaces the text with the text of the statements in the macro definition.

The listing of the macro **OUTPUT** is given in APPENDIX A. This macro can write 1 to 4 sequential bytes to a single port. An example of the macro used to output 2 bytes to a port is the instruction **'OUTPUT CTRL3, 9, 8'** as seen in the **SNGFRQ** listing. This macro outputs in sequence the two values 9 and 8 to section 3 control port. These are the 8255A commands to set and reset bit C4 of port C.

4.2.2 MULTIPLE FREQUENCY -- MLTFRQ

The assembly routine **MLTFRQ** is like the routine **SNGFRQ** executing in a loop. **MLTFRQ** receives as input the number of frequency values from the table of frequencies, the table memory address pointer, and the number of time the table is scanned.

The following operations are implemented in **MLTFRQ**:

a. the scan loop parameters are calculated;

b. the frequency table is scanned (BCD numbers are converted and output to the synthesizer, and the synthesizer is strobed); and

c. the keyboard input buffer is checked for a character.

If the input buffer is empty the zero flag is set and **MLTFRQ** restart the scanning. If the input buffer is not empty the routine returns and the menu reappears on the screen.

4.2.3 FREQUENCY HOPPING INTERRUPT CONTROLLED

The FORTRAN routine **FHINT** is called when interrupt control is selected. The frequency hopping rate and timing is controlled by an external hop clock signal which interrupts the computer. At each interruption, control is passed to an ISR which sends a strobe signal to the synthesizer, generates a BCD number taken from a table of frequency, and outputs the BCD formatted frequency value to the synthesizer. This process is controlled by **FHINT** which calls the assembly language routine **SUINT** to setup the interrupt mechanism and introduce the ISR to the operating system by changing the interrupt vector table. The interrupt utility subroutines, which are part of the **SUINT** module and used by **FHINT**, are described at the end of this sub-section.

4.2.3.1 **Routine FHINT.** The FORTRAN routine **FHINT** calls the assembly routine **SUINT** to setup the interrupt mechanism in the DOS operating system and sets the Base Board Section 3 PPI to mode 1 output (PPI command #0A4). As group A lines are used for the interrupt, the interrupt enable bit $INTE_A$ (bit C6) is set. This starts the process because when $INTE_A$ is set the $INTR_A$ line of the PPI, which is connected to IRQ7, sends an interrupt to the computer on the raising edge of the hop clock. The hop clock is connected to the ACK_A input line of the PPI. Detailed explanations of this mechanism may be found in [1].

The program goes into a wait loop checking only the interrupt count. This count is incremented inside the ISR each time a complete frequency table scan is done. When the table scan count is reached (see 2.4.3.1), **KBCHK** checks the input keyboard buffer for a character. If the buffer is empty, the count is zeroed (**ZCNT**) and the table scanning loop continues. If a character is entered by the user, the program disables interrupts, resets the old interrupt vector and IMR (**DISINT**), and returns to main program.

4.2.3.2 **Routine SUINT.** The routine **SUINT** starts by doing data initialization and setting up a new vector in the interrupt vector table for vector 0FH which is logically connected to the IRQ7 interrupt line. The old vector is saved for later restoring (**DISINT**) when this frequency hopping mode terminates. The IMR is then changed to unmask IRQ7 interrupts. Before returning to **FHINT**, **SUINT** sends a specific EOI to the 8259A PIC to allow pending level 7 interrupt to be serviced. After exiting **SUINT** the computer may be interrupted as soon as **FHINT** sets the $INTE_A$ bit.

4.2.3.3 **Interrupt Service Routine.** The ISR is part of the **SUINT** module. The routine is given control by the computer each time an interrupt signal is sensed on the IRQ7 line. Its function is to accomplish the following operations:

a. strobe the COMSTRON synthesizer to generate the frequency whose BCD value is stored at its input and was prepared during the previous call to this ISR;

b. fetch the next frequency value from the table of frequencies and convert it to BCD; and

c. output the new BCD value to the synthesizer.

The synthesizer will latch to the new frequency at the next interrupt. The ISR will also increment an interrupt count after each completed table scan. This count is continuously checked (**TBLCNT**) by the idling **FHINT** routine as a way to control the termination of this frequency hopping mode.

The main part of the ISR is coded similarly to **SNGFRQ** except for the fact that the synthesizer strobing is done before processing the frequency. At its beginning the ISR saves the computer environment, strobes the synthesizer and initializes the registers used to get the frequency from the table. BCD conversion and output to the synthesizer follow. Finally, the table parameters are adjusted for subsequent table access at the next interrupt. A check is made for the end of table scan and the interrupt count is incremented each time a complete table scan is done. Before exiting the ISR (instruction IRET) an EOI is output and the environment is restored.

An optional pulse signal may also be sent at the end to sense the instant the ISR is ready to terminate. This pulse, detected at test point TP-1 and observed on a digital signal analyzer along with the $INTR_A$ at TP-3 and the strobe pulse, was used to determine the duration of the ISR execution. To enable this optional pulse an instruction must be added at label **fin:** in the ISR. The following 'commented' instruction in the **SUINT** module must be modified:

```
fin:
;       OUTPUT          CTRL3, 0bH, 0ah    ;trace pulse at TP-1.
```

The comment operator (;) must be removed to enable the pulse to be generated. This will require a recompilation of the **SUINT** module and a relinking of the **SYNTCTR** program.

4.2.3.4 **Interrupt Utility Routines in Module SUINT.** The following three assembly utility routines in the **SUINT** module are used by **FHINT** for the management of interrupt service. They are:

a. **TBLCNT** -- used to get the interrupt table scanning count from the **SUINT** data area;

b. **ZCNT** -- called to zero that count before restarting another frequency table scanning; and

c. **DISINT** -- used to disable the interrupt processing. The $INTE_A$ bit is reset to prevent the 8255A PPI $INTR_A$ to send an interrupt signal through the IRQ7 line. The interrupt level-7 bit is masked (through IMR) to prevent interrupt from the IRQ7 line, and the old 0FH vector is restored by installing the old IRQ7 handler.

4.3 SECONDARY SUBROUTINES

Three other utilities are used in the **SYNTCTR** program, they are **DOUT**, **KBCHK** and **WAIT**.

4.3.1 DOUT

DOUT is an assembly subroutine which outputs to a PC port a byte value. It is used to send commands to the 8255A PPI.

4.3.2 KBCHK

KBCHK is an assembly subroutine used to read the contents of the keyboard input buffer. A return code and a character, if it exists, are returned through the calling arguments. This function is implemented using the DOS 06H Direct Console I/O function of software interrupt system call 21H. A return code of value 0, 1 or 2 is always returned. Zero is for no character in the buffer, one for 1 character, and two for a special character. The special characters are the extended characters that can be produced by keys such as ALT-O and F8. A table of these special codes can be found in the IBM PC Technical Reference Manual [5].

4.3.3 WAIT

WAIT is a FORTRAN subroutine designed to idle the program for a number of hundreds of seconds. This number is the call argument of **WAIT**.

4.4 PROGRAM MAINTENANCE

The eight software modules described above have been compiled with the Microsoft FORTRAN 4.1 and the Macro Assembler 5.1 compilers. The object (.OBJ) files resulting from these compilations were combined into a single executable (.EXE) file by the Microsoft linker LINK. The three modules written in FORTRAN are **SYNTCTR**, **FHINT** and **WAIT**. The remaining five routines written in Macro Assembler are **SNGFRQ**, **MLTFRQ**, **SUINT**, **DOUT** and **KBCHK**.

During the development of this controller several versions of the software modules were written to try different programming techniques or different synthesizers. To keep the changes to a minimum the calling sequence did not change, only the filenames. Fig. 4.1 lists the final version of the software

modules along with their filenames. The CS ending in the filename stands for COMSTRON synthesizer version.

MODULES	SOURCE FILENAMES
SYNCTR	SYNTCTR.FOR
SNGFRQ	SNGFRQCS.ASM
MLTFRQ	MLTFRQCS.ASM
FHINT	FHINTCS.FOR
SUINT	SUINTCS.ASM
DOUT	DOUT.ASM
WAIT	WAIT.FOR
KBCHK	KBCHK.ASM

Fig. 4.2 Filenames of software modules

When compiling the source programs the two compilers operates differently. The Macro Assembler will compile only one source module at a time (.ASM file). The FORTRAN compiler on the other hand can compile and/or compile and link source (.FOR) and object files or simply link object files and produce an executable file. Representative examples of the commands follow.

```
MASM /c/l/v/z/mx SNGFRQ ;                                    (MA.BAT)

FL /c/Fs/FPi87/G2 SYNTCTR.FOR, FHINTCS.FOR                   (FC.BAT)

LINK  SYNTCTR+FHINTCS+SNGFRQ+MLTFRQ+WAIT+KBCHK+SUINTCS+DOUT;
                                                             (LSCS.BAT)
```

The first example shows the command for a macro assembler compilation. The (;) at the end of the command tells MASM to use the default filenames for the object and listing (.LST) files.

The next example is a FORTRAN compiler command. This command request the compilation of two routines and the production of their respective object and listing files. The extension .FOR is required in the FORTRAN compiler command.

In the third example, the command line causes LINK to load and link all the object modules of the synthesizer controller. The executable file produced is **SYNTCTR.EXE**. In the LINK command a space or a plus sign (+) must separate each object filename. The .OBJ extension of the filenames however is not required in the command line.

The names in parenthesis on the right side in the examples block are the name of the batch (.BAT) files used to execute the commands. A batch file is a text file containing a list of commands to be executed, it has always a .BAT extension. The previous examples could have been written this way:

MA SNGFRQ

FC SYNTCTR.FOR FHINTCS.FOR

LSCS

Only one file may be the argument of MA.BAT. But up to nine files may be used with FC.bat.

To facilitate program development which includes editing, compiling, linking and debugging, the files have been arranged in separate directories according to their functions. DOS environment variables are used to specify directory path or locations and default options for library, macro and executable files. A batch file **PROGENV.BAT**, listed in APPENDIX A, defines the following environment variables: PATH, LIB, MASM, LINK, INCLUDE, and INIT which are described in [6] and [7].

Before starting any work for updating or maintaining the system the **PROGENV** batch file must be run to setup the DOS environment for this purpose as shown below:

C:\GTS\> PROGENV .

5.0 CONCLUSION

A frequency synthesizer controller for the purpose of testing the frequency hopping sub-system of the in-house FASSET ground terminal simulator has been presented in this paper. The user interaction with the controller was described showing how to install the hardware and how to run the software. The equipment, the controller interface to the synthesizer, and the PC's interrupt mechanism were described at the technical level. A description of the computer program and its modules was given at the programmer's level.

The hopping rate of the synthesizer in both the free running mode and the interrupt controlled mode was measured using a logic analyzer. The difference in performance between the two modes of operation was the free running rate being about double the interrupt controlled rate. This condition was explained by the computer and software overhead required to process an interrupt. To improve the performance of the controller, modification to the ISR is required to shorten its execution time. This can be achieved by calculating the BCD frequencies ahead of time outside the ISR and by not saving and restoring the floating point environment at the entry and exit of the ISR.

The synthesizer controller cannot be used to control the synthesizer of the simulator in real time because it is too slow. In addition the program used frequency values from tables. In real time simulation, each frequency value must be read from the PN-sequence generator, processed, corrected, etc. However, the system could be used to test the frequency hopping subsystem of the ground terminal simulator during its construction stage.

REFERENCES

[1]. BaseBoard™ Installation Manual Users Guide, Scientific Solutions, Inc.

[2]. Operation and Maintenance Manual, Series FS2000 Frequency Synthesizer, Comstron Corporation.

[3]. The MS-DOS Encyclopedia, Microsoft Press, Redmond, Washington, 1988.

[4]. Interrupts and the IBM PC Part 1 and Part 2, PC Tech Journal, Nov/Dec 1983 and January 1984.

[5]. Technical Reference Personal Computer AT, 6280070, IBM Personal Computer Hardware Reference Library.

[6]. Microsoft Macro Assembler 5.1, Programming Guide, pages 7-10.

[7]. Microsoft FORTRAN 4.1 Optimizing Compiler, Users Guide, pages 34-39.

APPENDIX A

SOFTWARE SOURCE CODE LISTINGS

A1 PROGRAM SYNTCTR

```
      PROGRAM SYNTCTR

C     Author:    C. Brochu
C     Filename:  SYNTCTR.FOR
C     Date:      May 19, 89

C     This program controls a frequency synthesizer using a menu
C     for the selection of the modes of operation which are:

C                        - Single frequency
C                        - Step frequency
C                        - Multiple Frequency or frequency hopping
C                          from a table of frequencies

C     This version of the program presents a selection for a
C     Frequency Hopping operation controlled by an interrupt clock
C     signal.

C     At interrupt time, the synthesizer is strobed to oscillate at
C     the frequency stored at its inputs, a new frequency value is
C     fetched from the table,  converted to BCD and outputed to the
C     synthesizer ready for the next clock pulse.

C     Interrupt is implemented using the IRQ7 interrupt line.

C     The interrupting clock signal is connected to the ~ACK A pin of
C     the 3rd section 8255 operating in Mode 1 (OUTPUT).  It is line
C     C6 of the 8255 or pin 10 of that section of the 96-bit digital
C     IO board.

C     Subroutines called:

C     MTLFRQ          MTLFRQCS.ASM
C     SNGFRQ          SNGFRQCS.ASM
C     FHINT           FHINTCS.FOR

C     DOUT            DOUT.ASM
C     WAIT            WAIT.FOR
C     KBCHK           KBCHK.ASM


C     ************************************************************************
```

```
      INTEGER*2 MODE, NOFRQ, I
      INTEGER*4 LPCNT, DWELL
      INTEGER*2 Badr96, CTRL1, CTRL3, MODE0

      REAL*8    TABLE (1000), FREQ, START, STOP, STEP, VAL, INC
      LOGICAL   SKIP

      CHARACTER*15 FNAME
      CHARACTER    ANS

c     Data initialization for
c     8255 ICs on 96-bit digital IO board

      Badr96 = #210
      CTRL1  = Badr96 + 0
      CTRL3  = Badr96 + 8
      MODE0  = #080


C     MAIN MENU

C     INITIALIZE THE DIGITAL IO BOARD (Mode 0) Sections 1 and 3

      CALL DOUT ( CTRL1, MODE0 )
      CALL DOUT ( CTRL3, MODE0 )

10    WRITE(*, 1000)
      READ(*, *) MODE
      GO TO (100, 200, 300, 400, 600), MODE
      GO TO 10

C     ------------------------------------------------------------------

C     SINGLE FREQUENCY MODE

100   WRITE(*, 1010)
      READ(*, *) FREQ
      IF( FREQ .LT. 0.0 ) GO TO 10
      CALL SNGFRQ ( FREQ )
      GO TO 100

C     ------------------------------------------------------------------

C     STEP FREQUENCY MODE

200   WRITE(*, 1050)
      READ(*, *) START, STOP, STEP, DWELL
      IF( START .LT. 0.0 ) GO TO 10

      SKIP = .TRUE.
      IF (DWELL .EQ. 0 ) SKIP = .FALSE.
```

```
      DO 250 FREQ = START, STOP, STEP

           CALL SNGFRQ ( FREQ )
           IF ( SKIP ) CALL WAIT ( DWELL )

c     Check keyboard buffer - any character, stop

           CALL KBCHK( ChRd, RtnCode )
           if ( RtnCode .ne. 0 ) go to 200

250   CONTINUE

      GO TO 200

C     ---------------------------------------------------------------

C     FREQUENCY HOPPED TABLE VALUES

300   VAL = 1.0
      INC = 4004004.0
      NOFRQ = 1000

      DO 350 I = 1, NOFRQ

           TABLE(I) = VAL
           VAL = VAL + INC

350   CONTINUE

      WRITE(*, 1100)

      GO TO 500

C     ---------------------------------------------------------------

C     FREQUENCY HOPPED SPECIFIC VALUES

400   WRITE(*, 1200)
      READ(*, '(A)') FNAME
      OPEN (1, FILE = FNAME, STATUS = 'OLD')
      READ(1, *) NOFRQ
      READ(1, *) (TABLE(I), I = 1, NOFRQ)

C     PROCESS THE FREQUENCY TABLE AND OUTPUT CONTROL
C     SIGNALS TO THE SYNTHESIZER

500   WRITE(*, 1125) NOFRQ
      WRITE(*, 1150)
      READ(*, *) LPCNT
      WRITE(*, 1175)

C     Query user to use Interrupt processing
```

```
      WRITE(*, 1110)
      READ(*,'(A)') ANS
      IF ( ANS .EQ. 'Y' .OR. ANS .EQ. 'y') go to 550

C     Scan table of frequencies

      CALL MLTFRQ ( NOFRQ, TABLE, LPCNT )

      go to 560

C     Scan table of frequencies  -  using interrupt processing

550   CALL FHINT ( NOFRQ, TABLE, LPCNT )

560   CLOSE(1)

      GO TO 10

600   STOP

C*********************************************************************

C     FORMATS STATEMENTS

C*********************************************************************


1000  FCRMAT(' ^[[2J'//
     *           T22,'D6470 - EHF GROUND TERMINAL PROJECT'/
     *           T22,'Synthesizer Interface Tests'//
     *           T22,'Mode    Operation'//
     *           T22,' 1    Single Frequency'/
     *           T22,' 2    Step   Frequency'/
     *           T22,' 3    Frequency Hopped - Table Values'/
     *           T22,' 4    Frequency Hopped - Specific Values'/
     *           T22,' 5    Exit'//
     *           T22,'Mode: '\)
1010  FORMAT(//' Enter Single Frequency ( Neg. for Menu )  > '\)
1050  FORMAT(//' Step Frequency Mode:  Please Enter 4 Numbers'//
     *           ' Start ( Neg. for Menu ), Stop, Step Frequency '
     *           'and Dwell time (100th of sec)'/' ? '\)
1100  FORMAT(//T24,'Frequency Hopped Table Values'/)
1110  FORMAT(' Do you whish to use the interrupt mechanism ( Y/N) ? '\)
1125  FORMAT(' There are,'I5,'  Frequency Values in Table')
1150  FORMAT(' Enter Table Scan Count > '\)
1175  FORMAT(/' Any Character will stop Table Scan ')
1200  FORMAT(//T23,'Frequency Hopped Specific Values'//
     *           ' Enter Specific Values  File Name > '\)


      END
```

```
A2    SUBROUTINE SNGFRQ

Title  Convert Frequency to BCD & OUT TO PORTS
SUBTTL COMSTRON Synthesizer Version

      PAGE ,132

;   *********************************
;    *
;     ** File Name: SNGFRQCS.ASM
;    *
;   *********************************

;      Author:     C. Brochu
;      Date:       Feb 3, 89

;      This routine converts the input frequency to BCD and output
;      BCD Values to the COMSTRON synthesizer via  the Scientific
;      Solutions 96-bits digital IO board.

;      10 BCD digits for frequencies ranging from 1Hz to 3.999999999GHz
;      are sent to ports

;              A1, B1, C1, A3, B3.

;      These values are taken from the Ten Byte variable
;      BCDVAL from offset 0 to 5 respectively.

;      NOTE:

;         - Bit C3-4 (C4 of Section 3) is used to strobe the COMSTRON
;           Synthesizer


;      On exit the Synthesizer output frequency is latched.

;      ____________________________________________________________


;     FORTRAN Calling Sequence

;           CALL SNGFRQ ( Freq )

;           Freq   Frequency Value (REAL*8)


;      ____________________________________________________________


.sall
.286
```

```
.Model Large, FORTRAN

Include GTS.INC


;       Equates

BCDVAD          EQU     BYTE PTR BCDVAL
DUMY            EQU     ES:[BX]

BaseADD         EQU     210H
PRTA1           EQU     BaseADD +  3
PRTB1           EQU     BaseADD +  2
PRTC1           EQU     BaseADD +  1
PRTA3           EQU     BaseADD + 11
PRTB3           EQU     BaseADD + 10
CTRL3           EQU     BaseADD +  8

;           Base address of 96-bit digital IO board:  210H

;           Ports          IO Location Value

;             A1               3
;             B1               2
;             C1               1
;             A3              11
;             B3              10


;       ---------------------------------------------------------------

.data
BCDVAL              DT      0               ;Ten byte for BCD value

;       ---------------------------------------------------------------

.code

SNGFRQ              PROC    FAR ,    FREQ: ptr

            LES     BX, FREQ                ;LOAD FREQ VALUE

            FLD     QWORD PTR DUMY          ;Into 80287 for
            FBSTP   BCDVAL                  ;BCD CONVERSION
            FWAIT

;  -----  Output bcd values to the COMSTRON Synthesizer

       OUTPUT       PRTA1, BCDVAD+[0]    ;1Hz    + 10Hz
       OUTPUT       PRTB1, BCDVAD+[1]    ;100Hz  + 1KHz
       OUTPUT       PRTC1, BCDVAD+[2]    ;10KHz  + 100KHz
```

```
        OUTPUT      PRTA3, BCDVAD+[3]    ;1MHz   + 10MHz
        OUTPUT      PRTB3, BCDVAD+[4]    ;100MHz + 1GHz

;       Strobe synthesizer, Latch frequency - Set/Reset bit C3-4
;       frequency change when neg. going edge occurs

        OUTPUT      CTRL3, 9, 8

                    RET                  ;RETURN

SNGFRQ              ENDP
                    END
```

```
A3    SUBROUTINE MLTFRQ

Title  OUTPUT FREQUENCY FROM A TABLE
SUBTTL COMSTRON Synthesizer Version

      PAGE ,132

;   **********************************
;    *
;     ** File Name: MLTFRQCS.ASM
;    *
;   **********************************


;      Author:     C. Brochu
;      Date:       Feb 27, 89


;      This routine reads frequencies from a TABLE, converts
;      them to BCD and output BCD Values to the COMSTRON synthesizer
;      via the Scientific Solutions 96-bits digital IO board.

;      10 BCD digits for frequencies ranging from 1Hz to 3.999999999GHz
;      are sent to ports

;              A1, B1, C1, A3, B3.

;      These values are taken from the Ten Byte variable
;      BCDVAL from offset 0 to 5 respectively.

;      NOTE:

;         - Bit C3-0 (C0 of Section 3) is used to strobe the COMSTRON
;           Synthesizer

;      ____________________________________________________________


;     FORTRAN Calling Sequence

;           CALL MLTFRQ ( N, TABLE, LOOPCTR )

;           Where:

;           N          No of BCD Frequencies in TABLE
;           TABLE      Frequency Value  (REAL*8)
;           LOOPCTR    No of time the table is scanned

;      ____________________________________________________________

.sall
.286
```

```
.Model Large, FORTRAN

Include GTS.INC

;       Equates

BCDVAD          EQU     BYTE PTR BCDVAL
TBLVAL          EQU     QWORD PTR ES:[BX]
DUMY            EQU     ES:[BX]

BaseADD         EQU     210H
PRTA1           EQU     BaseADD +  3
PRTB1           EQU     BaseADD +  2
PRTC1           EQU     BaseADD +  1
PRTA3           EQU     BaseADD + 11
PRTB3           EQU     BaseADD + 10
CTRL3           EQU     BaseADD +  8

;           Base address of 96-bit digital IO board:  210H

;         Ports    IO Location Value

;           A1          3
;           B1          2
;           C1          1
;           A3         11
;           B3         10

;     ------------------------------------------------------------------

.data
BCDVAL          DT      0               ;Ten byte for BCD value

;     ------------------------------------------------------------------

.code

MLTFRQ          PROC    FAR  USES DI , N: ptr, TABLE: ptr, LOOPCTR: ptr


MAINLP:             LES     BX, LOOPCTR         ;Setup TABLE loop counter
                MOV     DI, DUMY            ;Using DI

TBLLP:          LES     BX, N                       ;Setup Freq loop for
                MOV     CX, DUMY            ;N  Freqencies USING CX
                LES     BX, TABLE           ;LOAD TABLE Address

FRQLP:
                FLD     TBLVAL              ;Table value converted
                FBSTP   BCDVAL              ;to BCD stored into BCDVAL
                FWAIT
```

```
;  -----  Output bcd values to the COMSTRON Synthesizer

       OUTPUT   PRTA1, BCDVAD+[0]     ;1Hz    + 10Hz
       OUTPUT   PRTB1, BCDVAD+[1]     ;100Hz  + 1KHz
       OUTPUT   PRTC1, BCDVAD+[2]     ;10KHz  + 100KHz
       OUTPUT   PRTA3, BCDVAD+[3]     ;1MHz   + 10MHz
       OUTPUT   PRTB3, BCDVAD+[4]     ;100MHz + 1GHz


;      Strobe synthesizer, Latch frequency - Set/Reset bit C3-4
;      frequency change when neg. going edge occurs

       OUTPUT   CTRL3, 9, 8

            ADD     BX, 8             ;Adj. Offset for next TABLE Value
            LOOP    FRQLP             ;Loop N times

            Dec     DI                ;Dec. Table Loop counter
            JNZ     TBLLP             ;Exit after N*LOOP Frequencies
;  -----
            MOV     AH, 06H           ;Check keyboard input buffer
            MOV     DL, 0FFH          ;Using
            DOSCALL                   ;Direct Console IO

            JZ      MAINLP            ;No input, Z Flg Set, do it again
                                      ;Input, terminate

            RET                       ;RETURN

MLTFRQ      ENDP
            END
```

```
A4    SUBROUTINE FHINT

c     Author:       C. Brochu
c     File name:    FHINTCS.FOR
c     Date:         Feb 14, 89


c     Device used:   8255 ICs from 96-bit digital IO board


c     This routine produce a frequency hopping system controlled
c     by a clock signal.  This signal will interrupt the computer,
c     by means of a 8255 IC operating in mode 1 (output).

c     When an interrupt occurs the synthesizer is strobed to latch
c     the frequency value stored at its input and a new frequency
c     is prepared for latching at the next clock.  ( This frequency
c     value is fetched from the TABLE, converted to BCD and output
c     to the synthesizer ).  This BCD value is ready to be latched
c     at the next clock pulse.


c          Calling Sequence

c          CALL FHINT ( NOFRQ, TABLE, LOOPCTR )

c          Where:

c          NOFRQ      No of Frequencies in TABLE
c          TABLE      Frequency - REAL Value
c          LOOPCTR    No of time the table is scanned


c          Routines Called:

c          DIN               Byte in from port
c          DOUT              Byte out to Port
c          DBOUT             Double byte out to Port
c          SUINT  (SUINTCS)  Set up new IRQ7 Vector for 8255 Int.
c          KBCHK             Check keyboard input  buffer
c          WAIT              Wait

c          TBLCNT (SUINTCS)  Get TABLE scan count
c          DISINT (SUINTCS)  Disable Interrupt and Restore
c                            the old Interrupt Vector OF
c          ZCNT   (SUINTCS)  Zero TABLE scan count

c     ***************************************************************
```

```
      SUBROUTINE FHINT ( NOFRQ, TABLE, LOOPCTR )

      integer*2  MODE0, MODE1, ENINTA
      integer*2  Badr96, ctrl1, ctrl3, nul
      integer*2  count, clkcnt, freq, ChRd, RtnCode
      integer*2  LOOPCTR, NOFRQ
      REAL       TABLE(1)

c     Data initialization for
c     8255 ICs on 96-bit digital IO board

      Badr96 = #210
      CTRL1  = Badr96 + 0
      CTRL3  = Badr96 + 8

      MODE0   =  #080
      MODE1   =  #0A4
      ENINTA  =  #0D
      NUL     =  0
      RtnCode =  0

c     Set up interrupt through IRQ7 and make the frequency array
c     parameters known to the interrupt service routine

      call SUINT ( NOFRQ, TABLE )
      write ( *, '(/5x,''Interrupt Mechanism Set Up''\)' )

c     Set up Section 3  8255 IC to MODE 1 digital output

      call DOUT ( ctrl3, mode1 )

c     Enable interrupt on Section 3  8255
c          -- SET C6 INTE-A bit

      write ( *, '('' -- Interrupt ENABLED'')' )
      call dout ( ctrl3, eninta )

c     Check TABLE scan count

20    call TBLCNT ( count )

      if ( count .lt. loopctr ) go to 20

c     Check keyboard buffer - any character, stop
c                - no char,       count is zeroed

c     call KBCHK( ChRd, RtnCode )
c     if ( RtnCode .eq. nul ) then
c          call ZCNT
c          go to 20
c     endif
```

```
c               TERMINATION

c       Disable interrupt --   Reset C6 INTE-A bit
c                   Reset INTR level
c       Mask Interrupt Level 7 and Reset old interrupt vector
c       Return to main MENU

        call DISINT

        write ( *, 1000)
        call WAIT (150)

c       ************************************************************************

1000    format(5x,'^GInterrupt DISABLED -- Interrupt Mechanism Removed')

        end
```

A5 SUBROUTINE SUINT

```
Title  Frequency Hopping Set Up
SUBTTL Interrupt Control - COMSTRON Synthesizer

       PAGE ,132

;           **********************************
;            *                                *
;             ** File Name: SUINTCS.ASM     *
;            *                                *
;           **********************************


;      Author:     C. Brochu
;      Date:       Feb 13, 89


;      This routine will set up interrupt control using the Section 3
;      8255 IC of the 96-bit digital IO board, operating in MODE 1
;      (output).  The computer interrupt is triggered by the IRQ7 line
;      and is processed through interrupt vector 0FH.

;      The frequency TABLE array address and the table size are passed to
;      the interrupt service routine (ISR) by the calling arguments.


;             FORTRAN Calling Sequence

;             CALL SUINT ( NOFRQ, TABLE )

;             NOFRQ       No of Frequencies in TABLE (max 1000)
;             TABLE       Frequency - REAL*8 Values

;      ------------------------------------------------------------------

.SALL
.286

.Model Large, FORTRAN

Include       GTS.INC

;      Equates

SetVec          equ    25H
GetVec          equ    35H
IRQ7Int         equ    0FH
Spec_EOI        equ    67H       ;Specific EOI for IRQ7 Int. level 7
EOI             equ    20H       ;Non specific EOI (end of Int.)
OCR             equ    20H       ;Loc. of 8259 Operational Control Reg.
```

```
IMR             equ     21H         ;Loc. of 8259 Interrupt Mask Reg.

Badr96          equ     210H        ;96-Bit Board
PRTA1           equ     Badr96 +  3
PRTB1           equ     Badr96 +  2
PRTC1           equ     Badr96 +  1
PRTA3           equ     Badr96 + 11
PRTB3           equ     Badr96 + 10
PRTC3           equ     Badr96 +  9
CTRL3           equ     Badr96 +  8

;           Base address of 96-bit digital IO board:  210H

;         Ports     IO Location Value

;           A1          3
;           B1          2
;           C1          1
;           A3         11
;           B3         10
;           C3          9

BCDVAD          equ     BYTE PTR BCDVAL
TBLVAL          equ     QWORD PTR [BX][DI]

Dumy            equ     <es:[bx]>       ;reference address of a dummy var.

;       ---------------------------------------------------------------

.DATA

OldVec0F        DD      0               ;old vector 0FH content
BCDVAL          DT      0               ;BCD value storage

tbladr          dw      0               ;address of table
tblsze          dw      0               ;its size in bytes
tbldspl         dw      0               ;byte displ-nt in table
tblcntr         dw      0               ;TABLE scan count
ISB             db      08h             ;Interrupt A status bit

PFPENV          db      28 dup (0)      ;Temporary storage for FP
PFPST0          dq      0               ;environment and ST(0)

;       ---------------------------------------------------------------

.code

SUINT       PROC    FAR     uses es,  NOFRQ: PTR,  TABLE: PTR


;  -----  Store Table address and size for future use
```

```
            les     bx, TABLE
            mov     tbladr, bx          ;store TABLE address

            les     bx, NOFRQ
            mov     ax, dumy
            mov     tblsze, ax          ;Multiply array size by 8
            shl     tblsze, 1
            shl     tblsze, 1           ;to get size in bytes
            shl     tblsze, 1           ;store result in tblsze

;   -----   Initialize some variables

            mov     tbldspl, 0
            mov     tblcntr, 0

;   -----   Get current content of IRQ7 vector (0FH)

            cli                         ;Interrupt off during set up
            mov     ah, GetVec
            mov     al, IRQ7Int
        DOSCALL
            mov     word ptr OldVec0F+2, es       ;save segment and offset
            mov     word ptr OldVec0F,   bx       ;of previous IRQ7 handler

;   -----   Set up Interrupt vector 0FH (for IRQ7) to point to   MOD1INT

            push    ds                  ;save DS
            push    cs                  ;set      DS = CS
            pop     ds                  ;  DS:DX = handler address
            mov     dx, offset MOD1INT
            mov     ah, SetVec          ; Install new handler
            mov     al, IRQ7Int
        DOSCALL                         ; Set new IRQ7 vector
            pop     ds                  ;restore DS

;   -----   Enable IRQ7 on 8259 Interrupt Controller

            in      al, IMR             ;read current IMR
            jmp     $+2                 ;IO delay allow time
            jmp     $+2                 ;between outputs to 8259A
            and     al, 07FH            ;unmask int. level 7
;           or      al, 01H             ;mask timer interrupt
            out     IMR, al             ;write new int. mask
                                ;issue specific EOI to allow pending
        OUTPUT OCR, Spec_EOI    ;level 7 interrupt to be serviced

            sti                         ;enable interrupt
            ret

SUINT       ENDP

;       ---------------------------------------------------------------
```

```
;       Interrupt service routine (ISR)

;       Interrrupt generation done via an 8255 IC operating in mode 1
;       (output)

;       This ISR will:

;            - strobe the COMSTRON Synthesizer to produce the
;          frequency loaded at its input and prepared during the
;          previous call to this ISR.

;            - fetch a frequency value from the TABLE and convert it to
;              BCD

;            - and output the new BCD value to the synthesizer

;       The synthesizer will latch to this frequency at the next
;       interrupt.

;       10 BCD digits for frequencies ranging from 1Hz to 3.999999999GHz
;       are sent to (8255's) ports  Section 1 & 3

;             A1, B1, C1, A3, B3.

;       These values are taken from the Ten Byte variable
;       BCDVAL from offset 0 to 5 respectively.
;
;       NOTE:

;          -  Bit C3-4 (C4 of Section 3) is used to strobe the COMSTRON
;             Synthesizer
;       ---------------------------------------------------------------

MOD1INT:
            pushf                       ;save all registers
            push    ax
            push    dx

;  -----  Test if Interrupt caused by the 96-bit board - 8255-3

        INPUT        PRTC3, al          ;Read status register
             test    al, ISB            ;Interrupt caused by 8255-3?
             jnz     Service            ;Yes - Service Interrupt

             pop     dx                 ;No - pass control
             pop     ax                 ;to former vector 0FH
             popf
             jmp     dword ptr cs:OldVec0F

;       ---------------------------------------------------------------
```

```
Service:
            push    bx                  ;save 1 more register
            fstp    PFPST0              ;and FP environment
            fstenv  PFPENV
            FWAIT

;       Strobe synthesizer, Latch frequency - Set/Reset bit C3-4
;       frequency change when neg. going edge occurs

        OUTPUT      CTRL3, 9, 8

;       Get frequency from TABLE, convert to BCD

            mov     bx, tbladr          ;Set up to address
            mov     di, tbldspl         ;table content

            FLD     TBLVAL              ;Table value converted
            FBSTP   BCDVAL              ;to BCD stored into BCDVAL
            FWAIT

;  -----  Output bcd values to the COMSTRON Synthesizer

        OUTPUT      PRTA1, BCDVAD+[0]       ;1Hz        + 10Hz
        OUTPUT      PRTB1, BCDVAD+[1]       ;100Hz  + 1KHz
        OUTPUT      PRTC1, BCDVAD+[2]       ;10KHz  + 100KHz
        OUTPUT      PRTA3, BCDVAD+[3]       ;1MHz   + 10MHz
        OUTPUT      PRTB3, BCDVAD+[4]       ;100MHz + 1GHz

;  -----  Writing to group A3 resets the Interrupt Request signal
;       INTR A (bit C3-3)

;       Add displacement for next 64-bit item in TABLE

            add     tbldspl, 8          ;adjust displacement for next
            mov     ax, tblsze          ;item in table
            cmp     tbldspl, ax         ;is it the end of the table?
            jle     fin                 ;no, continue
            mov     tbldspl, 0          ;yes, restart at top of table

            inc     tblcntr             ;increment TABLE scan count

;  -----  Exit from ISR  --  Signal EOI to 8259 interrupt controller

fin:
;       OUTPUT      CTRL3, ObH, Oah     ;trace pulse at TP-1
;           cli                         ;no interrupt allowed

        OUTPUT OCR, EOI                 ;Non-specific EOI to 8259

            fldenv  PFPENV              ;restore registers
            fld     PFPST0
            pop     bx
```

```
            pop     dx
            pop     ax
            popf

            iret                        ;ISR return

;       ---------------------------------------------------------------

;       Fortran callable routine to disable interrupt on the digital
;       IO board, mask IRQ7 interrupt and reset old IRQ7 vector

;           FORTRAN Calling Sequence

;               CALL DISINT


DISINT          PROC    FAR     uses DS

            cli                         ;no interrupt allowed

;       Disable interrupt on 8255-3 (Section 3 of 96-bit board)
;       Reset INTR A level by wrinting to Group A
;       in case of a pending interrupt request
;       Set Section 3 to Mode 0

        OUTPUT      CTRL3, 0CH          ;Reset bit C6 (INTE A)
        OUTPUT      PRTA3, BCDVAD+[3]   ;Reset INTR
        OUTPUT      CTRL3, 0CH          ;Set to Mode 0

;   -----  Mask interrupt level 7  -  Disable IRQ7 on 8259

            in      al, IMR             ;read current IMR
            jmp     $+2                 ;IO delay allow time
            jmp     $+2                 ;between outputs to 8259A
            or      al, 80H             ;mask int. level 7
            out     IMR, al             ;write new int. mask

;   -----  Install old IRQ7 handler

            lds     dx, OldVec0F        ;DS:DX = handler address
            mov     ah, SetVec          ; Install old handler
            mov     al, IRQ7Int
        DOSCALL
                                        ;issue specific EOI to allow
        OUTPUT OCR, Spec_EOI            ;pending level 7 interrupt
                                        ;to be serviced
            sti                         ;enable interrupt
            ret

DISINT      ENDP

;       ---------------------------------------------------------------
```

```
;       Fortran callable routine to get TABLE scanning count

;           FORTRAN Calling Sequence

;           CALL TBLCNT( COUNT )

;              COUNT        Int*2

TBLCNT          PROC   FAR    uses ds,  count: ptr

            les    bx, count          ;count address
            mov    ax, tblcntr        ;get value
            mov    Dumy, ax           ;store value into count

            RET                 ;RETURN

TBLCNT          ENDP

;       ------------------------------------------------------------

;       Fortran callable routine to zero the interrupt
;       table loop through count

;           FORTRAN Calling Sequence

;           CALL ZCNT

ZCNT        PROC   FAR

            mov    tblcntr, 0      ;zero value
            ret

ZCNT        ENDP

            END
```

A6 SUBROUTINE DOUT

```
Title   Digital Output General Routine

          PAGE , 132

;           **********************************
;            *                                *
;             ** File Name: DOUT.ASM         *
;            *                                *
;           **********************************


;       Author:     C. Brochu
;       Date:       May 19, 88

;      This general routine will output to
;                a PC Port       (ARG1)
;                a Byte Value    (ARG2)


;      FORTRAN Calling Sequence

;          CALL DOUT ( PCPort, BValue)

;            PCPort       Port address

;            BValue       Byte value to output


.lall
.Model Large, FORTRAN
.code

DOUT   PROC  FAR, IOPort:PTR, BytVal:PTR

       LES BX, IOPort
       MOV DX, ES:[BX]    ;LOAD PORT ADDR IN DX

       LES BX, BytVal
       MOV AL, ES:[BX]    ;LOAD BYTE VALUE IN AL

       OUT DX, AL         ;WRITE TO PC PORT
       MOV AH, 0

       RET       ;RETURN, POP 8 BYTES

DOUT   ENDP
       END
```

A7 SUBROUTINE KBCHK

```
Title Check Keyboard Input
Subttl If Any

        PAGE ,132

;           ***********************************
;            *
;             ** File Name: KBCHK.ASM
;            *
;           ***********************************


;       Author:     C. Brochu
;       Date:       Oct 12, 88


;       This routine will check the keyboard input buffer and will output
;       output a return code and the character if it exist.

;       The return codes (RtnCode) are:

;            0      -       No character in buffer
;            1      -       1 character in buffer, value in  ChRd
;            2      -       A double character in buffer, value in  ChRd
;                           (see table in PC Tech. Ref. Chapter 5)


;            FORTRAN Calling Sequence

;                CALL KBCHK ( ChRd, KbST )

;       ------------------------------------------------------------------


.Model Large, FORTRAN

include GTS.INC

.SALL

Dumy        equ     <es:[bx]>


.code


KBCHK       PROC    FAR     , ChRd: ptr, RtnCode: ptr

            mov     cl, 0               ;Init Return code
```

```
                mov     ah, 6               ;direct console IO
                mov     dl, 0ffH            ;INPUT

                jmp     read
DblChr:           inc     cl                ;increment return code

read:   DOSCALL

                jz      exit                ;exit if no char read
                test    al, 0ffH            ;check for double char

                jz      DblChr

                inc     cl                  ;increment return code
                les     bx, ChRd
                mov     Dumy, al            ;Store character read

exit:           les     bx, RtnCode
                mov     Dumy, cl            ;Store return code

                ret                         ;RETURN

KBCHK           ENDP
                END
```

A8 SUBROUTINE WAIT

```
C       Author:     C. Brochu
C       Filename:   WAIT.FOR
C       Date:       May 19, 89


        SUBROUTINE WAIT (HSEC)

C       This routine will loop for approximatively
C       HSEC Hundreds of seconds

        INTEGER*4 HSEC, CURTIM, TIMECHK, TIM
        INTEGER*2 HR, MN, SEC, HDS

        TIM(HR,MN,SEC,HDS) = HR*360000 + MN*6000 + SEC*100 + HDS

        CALL GETTIM (HR, MN, SEC, HDS)

        TIMECHK = TIM(HR,MN,SEC,HDS) + HSEC

10      CALL GETTIM (HR, MN, SEC, HDS)
        CURTIM = TIM(HR,MN,SEC,HDS)

        IF ( CURTIM .LT. TIMECHK ) GO TO 10

20      RETURN
        END
```

A9 BATCH FILES LISTING

```
@ECHO OFF
REM
REM    BATCH FILE TO SET UP PROGRAMMING ENVIRONMENT
REM    PROGENV.BAT
REM
PATH C:\PROG\EDIT\M;C:\PROG\MSFOR;C:\PROG\MASM\BIN;C:\SYSTEM\BAT
SET INIT=C:\PROG\MASM\BIN
SET LIB=C:\PROG\MSFOR
SET MASM=/c/l/v/z/mx
SET LINK=/m
SET INCLUDE=C:\PROG\MASM\INCLUDE


@REM    FORTRAN COMPILER BATCH FILE
@REM    FC.BAT
@REM
FL /c /Fs /FPi87 /G2 %1 %2 %3 %4 %5 %6 %7 %8 %9


@REM    MACRO ASSEMBLER BATCH FILE
@REM    MA.BAT
@REM
C:\PROG\MASM\BIN\MASM %1,,;


@REM    SYNTHESIZER CONTROL SYSTEM LINK BATCH FILE
@REM    LSCS.BAT
@REM
C:\PROG\MSFOR\LINK SYNTCTR+MLTFRQCS+SNGFRQCS+DOUT+WAIT+KBCHK+FHINTCS+SUINTCS
```

A10 INCLUDE FILE LISTING

```
;       Author:     C. Brochu
;       Filename:   GTS.INC
;       Date:       Sep 29, 88

;  Interface Macros -  for Microsoft Macro Assembler 5.0

.XLIST
.XCREF
IF1

OUTPUT          MACRO   PORT, VALUE1, VALUE2, VALUE3, VALUE4

            MOV     DX, PORT

                _outp VALUE1,VALUE2,VALUE3,VALUE4

            ENDM

_outp       MACRO  VALUE1,VALUE2,VALUE3,VALUE4

                IFNB   <VALUE1>
            MOV     AL, VALUE1
            OUT     DX, AL

                _outp VALUE2,VALUE3,VALUE4
                ENDIF

              ENDM


INPUT       MACRO  PORT, VALUE

            MOV     DX, PORT
            IN      AL, DX
            MOV     VALUE, AL

            ENDM


DOSCALL       MACRO
            int     21h
            ENDM

ENDIF                                   ; Pass 1
.LIST
```

APPENDIX B

WIRE-WRAP CONNECTIONS FOR COMSTRON SYNTHESIZER CONTROL INTERFACE

PARALLEL PORT 8255 SECTION 1			DRIVER 74LS244			COMSTRON SYNTHESIZER CONNECTOR	
DATA	PINS		PINS	PINS		PINS	DATA
A0	21	<-->	2	18	<-->	36	1 Hz-1
A1	22	<-->	4	16	<-->	11	-2
A2	23	<-->	6	14	<-->	35	-4
A3	24	<-->	8	12	<-->	10	-8
A4	25	<-->	11	9	<-->	19	10 Hz-1
A5	26	<-->	13	7	<-->	44	-2
A6	27	<-->	15	5	<-->	43	-4
A7	28	<-->	17	3	<-->	18	-8
			Vcc	20	--> + Post		
			Grnd	10	--> - Post		
			EN-	1,19	--> Grnd		
			U2				
B0	13	<-->	2	18	<-->	23	100 Hz-1
B1	14	<-->	4	16	<-->	48	-2
B2	15	<-->	6	14	<-->	22	-4
B3	16	<-->	8	12	<-->	49	-8
B4	17	<-->	11	9	<-->	27	1 KHz-1
B5	18	<-->	13	7	<-->	26	-2
B6	19	<-->	15	5	<-->	28	-4
B7	20	<-->	17	3	<-->	29	-8
			Vcc	20	--> + Post		
			Grnd	10	--> - Post		
			EN-	1,19	--> Grnd		
			U1				
C0	11	<-->	2	18	<-->	3	10 KHz-1
C1	31	<-->	4	16	<-->	2	-2
C2	12	<-->	6	14	<-->	4	-4
C3	32	<-->	8	12	<-->	5	-8
C4	9	<-->	11	9	<-->	34	100 KHz-1
C5	29	<-->	13	7	<-->	33	-2
C6	10	<-->	15	5	<-->	8	-4
C7	30	<-->	17	3	<-->	9	-8
+5V	3,38	--> + Post	Vcc	20	--> + Post		
Grnd	7,34	--> - Post	Grnd	10	--> - Post		
			EN-	1,19	--> Grnd		
J1			U3			J3	

WIRE-WRAP CONNECTIONS FOR COMSTRON SYNTHESIZER CONTROL INTERFACE

PARALLEL PORT 8255 SECTION 3			DRIVER 74LS244			COMSTRON SYNTHESIZER CONNECTOR	
DATA	PINS		PINS	PINS		PINS	DATA
A0	21	<-->	2	18	<-->	40	1 MHz-1
A1	22	<-->	4	16	<-->	39	-2
A2	23	<-->	6	14	<-->	41	-4
A3	24	<-->	8	12	<-->	42	-8
A4	25	<-->	11	9	<-->	46	10 MHz-1
A5	26	<-->	13	7	<-->	20	-2
A6	27	<-->	15	5	<-->	21	-4
A7	28	<-->	17	3	<-->	47	-8
			Vcc	20	--> + Post		
			Grnd	10	--> - Post		
			EN~	1,19	--> Grnd		
			U4				
B0	13	<-->	2	18	<-->	45	100 MHz-1
B1	14	<-->	4	16	<-->	17	-2
B2	15	<-->	6	14	<-->	16	-4
B3	16	<-->	8	12	<-->	15	-8
B4	17	<-->	11	9	<-->	13	1 GHz-1
B5	18	<-->	13	7	<-->	24	-2
C4	9	<-->	17	3	<-->	14	Strobe
C6	10	<-->	5	15	--> U6-6 IntClk		
C5	29	--> TP-1			(Coax Input)		
C7	30	--> TP-2					
C3	32	--> TP-3					
+5V	3,38	--> + Post	Vcc	20	--> + Post		
Grnd	7,34	--> - Post	Grnd	10	--> - Post		
			EN~	1,19	--> Grnd		
J2			U5				

PINS	PINS				PINS	
2 <--	1	1KΩ	4	<-->	38	1 GHz-4 --> 50
3 <--	2	1KΩ	5	<-->	37	1 GHz-8
- Post <--	3	IntClk	6	--> U5 15	30	10 Ghz-1
				+ Coax Input	32	.1 Hz-1 --> 50
					7	.1 Hz-2 --> 25
					31	.1 Hz-4 --> 50
					6	.1 Hz-8 --> 25
					25	Ground --> - Post
					50	Ground --> - Post
					J3	

DOCUMENT CONTROL DATA

(Security classification of title, body of abstract and indexing annotation must be entered when the overall document is classified)

1. ORIGINATOR (the name and address of the organization preparing the document. Organizations for whom the document was prepared, e.g. Establishment sponsoring a contractor's report, or tasking agency, are entered in section 8.)

Defence Research Establishment Ottawa
Ottawa, Ontario
K1A 0Z4

2. SECURITY CLASSIFICATION (overall security classification of the document including special warning terms if applicable)

UNCLASSIFIED

3. TITLE (the complete document title as indicated on the title page. Its classification should be indicated by the appropriate abbreviation (S,C or U) in parentheses after the title.)

A Frequency Hopping Synthesizer Controller for EHF SATCOM (U)

4. AUTHORS (Last name, first name, middle initial)

Brochu, Claude

5. DATE OF PUBLICATION (month and year of publication of document)

July 1989

6a. NO. OF PAGES (total containing information. Include Annexes, Appendices, etc.)

61

6b. NO. OF REFS (total cited in document)

7

7. DESCRIPTIVE NOTES (the category of the document, e.g. technical report, technical note or memorandum. If appropriate, enter the type of report, e.g. interim, progress, summary, annual or final. Give the inclusive dates when a specific reporting period is covered.)

Technical Note

8. SPONSORING ACTIVITY (the name of the department project office or laboratory sponsoring the research and development. Include the address.)

PM EHF SATCOM
Defence Research Establishment Ottawa
Ottawa, Ontario, K1A 0Z4

9a. PROJECT OR GRANT NO. (if appropriate, the applicable research and development project or grant number under which the document was written. Please specify whether project or grant)

D6470-0459B

9b. CONTRACT NO. (if appropriate, the applicable number under which the document was written)

10a. ORIGINATOR'S DOCUMENT NUMBER (the official document number by which the document is identified by the originating activity. This number must be unique to this document.)

DREO Technical Note 89-29

10b. OTHER DOCUMENT NOS. (Any other numbers which may be assigned this document either by the originator or by the sponsor)

11. DOCUMENT AVAILABILITY (any limitations on further dissemination of the document, other than those imposed by security classification)

(XX) Unlimited distribution
( ) Distribution limited to defence departments and defence contractors; further distribution only as approved
( ) Distribution limited to defence departments and Canadian defence contractors; further distribution only as approved
( ) Distribution limited to government departments and agencies; further distribution only as approved
( ) Distribution limited to defence departments; further distribution only as approved
( ) Other (please specify):

12. DOCUMENT ANNOUNCEMENT (any limitation to the bibliographic announcement of this document. This will normally correspond to the Document Availability (11). However, where further distribution (beyond the audience specified in 11) is possible, a wider announcement audience may be selected.)

UNCLASSIFIED

SECURITY CLASSIFICATION OF FORM

13. ABSTRACT (a brief and factual summary of the document. It may also appear elsewhere in the body of the document itself. It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall begin with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified) represented as (S), (C), or (U). It is not necessary to include here abstracts in both offical languages unless the text is bilingual).

ABSTRACT

The Ground Terminal Simulator is one of the in-house test facilities being developed for the system and technology evaluation phase of the FASSET project. A synthesizer controller has been developed using a personal computer to test the frequency hopping subsystem of the Ground Terminal Simulator. Hardware and software components of the test system are described at different levels of complexity; the level required for a user to utilize the system and the technical level required to understand and modify the hardware and to upgrade the software.

RESUME

Le simulateur de station terrestre de satellite est une des installations interne de tests en cours de dévelopment pour la phase d'évaluation de technologie et de système de FASSET. Un système de commande pour un synthétiseur de fréquence a été dévéloppé pour faire la vérification de sous-système de bond de fréquence du simulateur. Le matériel et le logiciel constituant ce système de vérification sont decrits à différents niveaux de complexités soit le niveau requis pour un utilisateur et celui requis pour maintenir et améliorer le materiel et le logiciel.

14. KEYWORDS, DESCRIPTORS or IDENTIFIERS (technically meaningful terms or short phrases that characterize a document and could be helpful in cataloguing the document. They should be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location may also be included. If possible keywords should be selected from a published thesaurus, e.g. Thesaurus of Engineering and Scientific Terms (TEST) and that thesaurus-identified. If it is not possible to select indexing terms which are Unclassified, the classification of each should be indicated as with the title.)

Frequency Hopping Test System

Frequency Hopping Synthesizer Controller

UNCLASSIFIED

SECURITY CLASSIFICATION OF FORM